



CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tél. : (1) 39 63 55 11

Rapports de Recherche

N° 467

**EVALUATION
OF RECURSIVE PREDICATS
IN DEDUCTIVE DATABASES**

Christophe de MAINDREVILLE

Décembre 1985

EVALUATION OF RECURSIVE PREDICATS
IN DEDUCTIVE DATABASES.

Christophe de Maindreville
SABRE Project, INRIA
BP 105, 78150 LE CHESNAY, France

Abstract :

This paper presents a survey of algorithms for computing recursive axioms in deductive databases. Different approaches are presented and analyzed with similar and characteristic examples. Finally further approaches are proposed.

LE TRAITEMENT DES AXIOMES RECURSIFS
DANS LES BASES DE DONNEES DEDUCTIVES.

Christophe de Maindreville

Projet SA3RE, INRIA

BP 105, 78150 LE CHESNAY, France

Résumé :

Cette étude présente une synthèse des diverses méthodes de contrôle de règles d'inférences récursives dans les SGBD déductifs.

Les différentes approches sont présentées puis discutées à l'aide d'exemples similaires et représentatif de leur spécificité.

De nouveaux axes d'études sont enfin proposés.

1. Introduction.

La mise en oeuvre des règles de déductions peut être considérée de deux manières différentes, [Nico83].

La première méthode consiste à les utiliser lors de la phase de mise à jour d'informations afin de générer puis de stocker des informations déductibles. Cette méthode est appelée méthode de génération.

Remarquons que la mise en oeuvre de cette méthode en présence de règles récursives ne pose pas de problèmes particuliers, [Mink83].

La seconde consiste à les mettre en oeuvre lors de la phase d'interrogation, afin de générer temporairement les informations nécessaires à l'évaluation de la requête. Cette méthode est appelée dérivation.

A la différence de la méthode précédente l'arrêt du processus d'inférence pose des problèmes de mise en oeuvre. En effet les techniques usuelles d'inférences utilisées en Intelligence Artificielle peuvent mener à parcourir des chemins de dérivation potentiellement infinis.

Dans le contexte des bases de données déductives, si les axiomes utilisés ne comportent pas de fonctions externes, "function-free axioms", le nombre de tuples inférés ne peut être que fini. Il est donc théoriquement possible de trouver des conditions d'arrêt pour tout chemin de dérivation.

Un second problème posé par la présence d'axiomes récursifs dans les bases de données déductives est la nécessité d'introduire de nouvelles techniques d'optimisation et d'évaluation de requêtes. En effet la

technique usuelle donnant la priorité aux sélections par rapport aux jointures se révèle inaplicable dans un contexte récursif.

2. Les méthodes compilées.

2.1 Introduction.

Le but de ces méthodes est de transformer une requête dont l'évaluation met en jeu un ensemble d'axiomes récursifs en un programme de requêtes n'invoquant que des relations explicitement stockées dans la base de données.

L'ensemble des prédicats appartenant à l'IDB (intentional database) sont donc compilés en expressions composées uniquement de prédicats appartenant à l'EDB (extensional database) et en une suite d'opérateurs de l'algèbre relationnelle. Les variables appartenant à ces expressions étant instanciées par les valeurs induites de la requête, le système de gestion de base de données peut alors évaluer cet ensemble de requêtes de manière tout à fait classique.

Cette approche permet donc de dissocier totalement les processus de résolution et d'évaluation des requêtes. Ceci autorise l'utilisation du processeur d'optimisation de questions des SGBD relationnels classiques, les accès à la base de données pouvant être différés et regroupés en un seul accès.

Ce type de méthode soulève principalement deux sortes de difficultés:

- l'obligation d'effectuer une optimisation statique des axiomes récursifs, ie, de profiter au maximum lors du processus de

résolution des instanciations procurées par la requête initiale afin de ne générer que les réponses désirées par l'utilisateur; ceci correspondant à effectuer au cours du processus d'évaluation les opérations de sélections le plus tôt possible.

La représentation de ces programmes itératifs au sein d'un langage hôte ou à l'aide d'opérateurs spécifiques du type opérateur de point fixe qui viendraient enrichir l'algèbre relationnelle classique ainsi que de leur structure de contrôle associée.

La plupart des méthodes que nous étudierons utilisent les techniques de chaînage arrière ou "goal-driven deduction" qui sont rappelées brièvement ici.

Soit P_i un prédicat apparaissant dans une requête $R: P_1 \& P_2 \& \dots \& P_i \& \dots \& P_m$. Supposons que ce prédicat ne puisse être résolu par aucune relation de l'EDB.

S'il existe une clause $C: Q_1 \& Q_2 \& \dots \& Q_n \rightarrow Q_{n+1}$ et une substitution s telle que $P_i.s = Q_{n+1}$ alors les prédicats P_i et Q_{n+1} sont dits unifiables. On peut alors remplacer P_i par le prémisses de la clause C pour obtenir une nouvelle requête,

$$R' : (P_1 \& P_2 \& \dots \& P_{i-1} \& Q_1 \& \dots \& Q_n \& P_{i+1} \& \dots \& P_m).s$$

telle que chaque solution de R' soit une solution de R .

Le processus est réitéré jusqu'à ce que tous les prédicats que comporte la requête puissent être résolus par les prédicats de l'EDB. C'est alors que s'effectue le processus d'évaluation sur les relations

explicitement stockées dans la base de données.

Cette technique de résolution peut conduire lors de l'évaluation des axiomes récurifs à parcourir des chemins de dérivation de longueur infinie. En effet le littéral non explicite étant réévalué à chaque étape de la récurion il faut pour éviter de "boucler" indéfiniment sur la meme requete que les arguments figurant dans ce littéral soient au moins partiellement instanciés afin de profiter des répercussions procurées par les prédicats appartenant à la base de faits. Le chainage arriere est donc inutilisable tel quel pour calculer par exemple la fermeture transitive d'une relation binaire.

De meme remarquons que la génération d'une suite d'expressions ne contenant que des relations de base ne permet pas toujours de choisir comme condition d'arret l'absence de résultats nouveaux à l'étape j du processus d'évaluation.

En effet il peut arriver qu'une expression d'ordre j+i puisse produire des tuples nouveaux.

Illustrons ceci par un exemple:

Exemple 1 :

Soient la relation ANCETRE définie par les axiomes suivants :

$PERE(x,y) \rightarrow ANCETRE(x,y)$

$PERE(x,y) \& ANCETRE(y,z) \rightarrow ANCETRE(x,z)$

ou la relation PERE est explicite, et la requete, $ANCETRE(a,b);$

Le développement de la définition récursive de la relation ANCETRE en relations explicites fournit alors la suite d'expressions suivante:

PERE(a,b)

PERE(a,x)&PERE(x,b)

PERE(a,x)&PERE(x,y)&PERE(y,b)

.....

Il est alors immédiat que l'absence de réponse à l'étape i ne permet pas de stopper définitivement le processus de recherche.

Pour remédier à ce genre de désagrément certains auteurs proposent d'alterner le chaînage arrière avec le chaînage avant, technique qui permet de "saturer" les relations virtuelles sur lesquelles porte la requête. Ceci permet donc de profiter des avantages procurés par ces deux méthodes:

- la seule génération des faits utiles pour la requête par le chaînage arrière.
- l'absence de problème de condition d'arrêt du processus de dérivation pour le chaînage avant étant entendu que l'on profite alors de la finitude du domaine sur lequel on procède à l'inférence.

2.2 Méthode de Chang.

Ces travaux [Chang81], font suite à [Chang78] où l'auteur proposait un langage d'interrogation de base de données relationnelle, DEDUCE2,

autorisant la manipulation d'informations sur des relations déduites. La méthode ne réglait pas le problème des vues définies récursivement.

Principe de la méthode:

Il s'agit de représenter l'ensemble forme de la requête et des axiomes impliqués en un graphe orienté, d'en déduire un ensemble de règles de transformation (rewriting rules) afin de générer une requête équivalente, compilable en un programme exécutable par le SGBD.

Cette méthode s'appuie sur deux concepts importants:

- le principe d'unification [Chang73].
- la théorie des langages réguliers [Kain].

Le principe d'unification permet de générer un graphe de connection [Kowa75] dont les arcs relient les littéraux figurant à droite d'une clause aux littéraux unifiables figurant à gauche.

Exemple 2 :

Soient les relations:

EMPLOYES(nom = x, manager = y, departement = z).

VENTES(departement = u, objet = v).

et la relation virtuelle COMMANDE(superieur = s, nom = t) définie par:

EMPLOYES(y, x) ---> COMMANDE(x, y).

EMPLOYES(z, y) & COMMANDE(x, y) ---> COMMANDE(x, z).

La relation COMMANDE est donc définie par un axiome de type récursif.

Soit la requete:

"Trouver toute les personnes qui sont commandées par jean et qui vendent des fleurs".

Celle-ci se représentera dans la syntaxe du langage DEDUCE2 par:

EMPLOYES(*nom = x, departement = y) &

VENTES(departement = y, objet = fleur) &

COMMANDE(superieur = jean, nom = x).

L'ensemble de la requête et des axiomes la concernant se représentera par le graphe suivant:

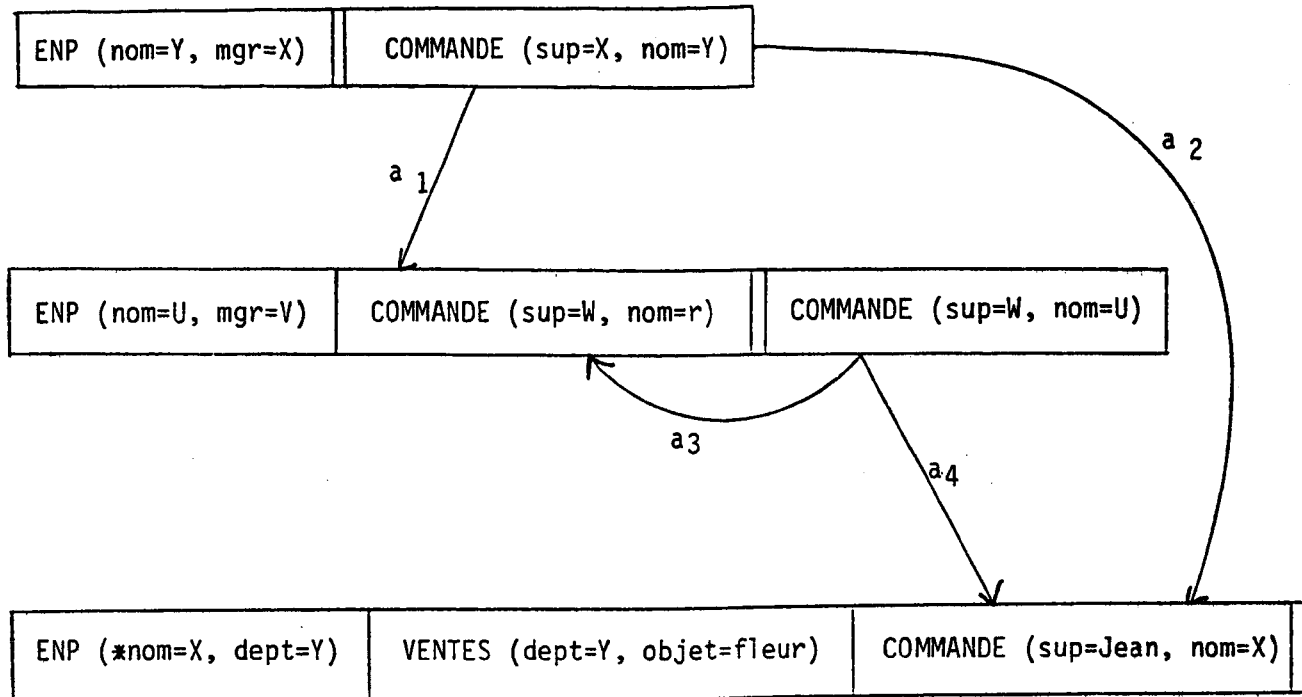


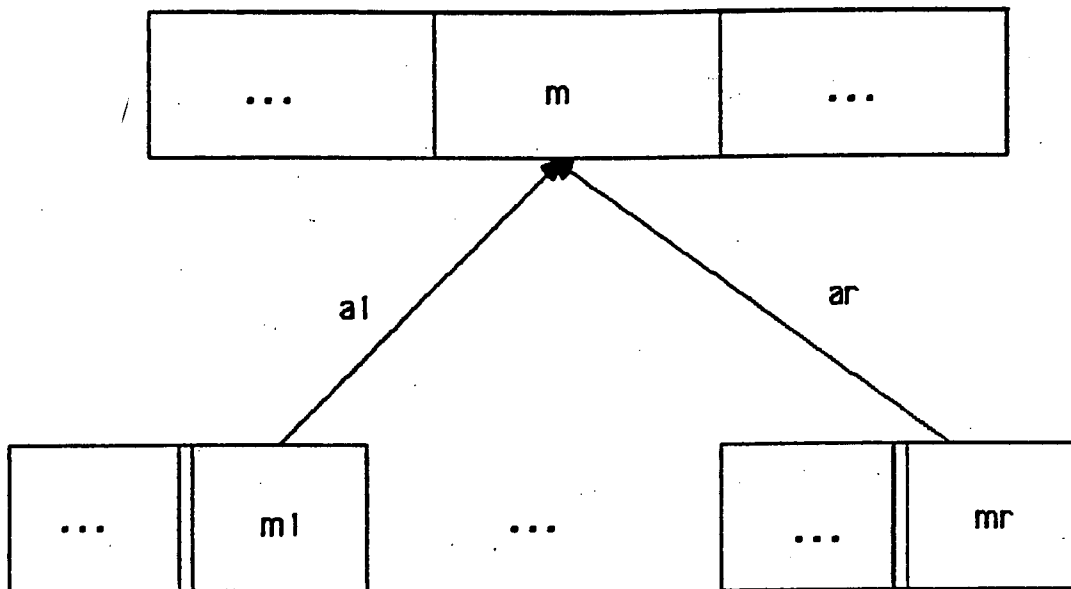
Schéma 1

Obtention des règles de transformations.

(i) pour les littéraux gauches: Soit n un littéral figurant à gauche d'une clause;

si m_1, \dots, m_r sont tous les littéraux droits pointant vers n on obtient alors la règle suivante:

$$W(n) = a_1 W(m_1) \cup \dots \cup a_r W(m_r).$$



Schema 2

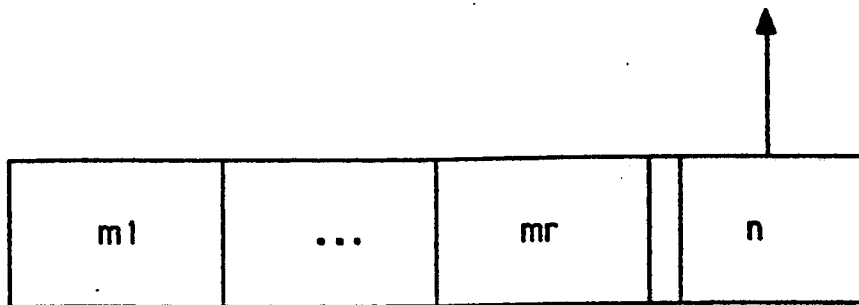
(ii) pour les littéraux droits:

Soit n un littéral figurant à droite d'une clause;

s'il existe un arc le quittant on obtient alors la règle :

$W(n) = P_1 \dots P_r$ ou les P_i sont définis par: $P_i = w(m_i)$ si m_i contient une relation virtuelle

= m_i sinon.



Schema 3

(iii) pour la requete:

Soit $R_1 \dots R_m$ une requete qui se représente par la clause suivante:

$R_1 \dots R_m \rightarrow$



Schema 4

La règle obtenue est alors la suivante:

$$R = Q_1 \dots Q_m$$

ou les Q_i sont définis comme suit: $Q_i = W(R_i)$ si R_i contient une relation virtuelle

$$= R_i \text{ sinon.}$$

On obtient ainsi une suite de règles exprimées à l'aide de symboles terminaux et non terminaux d'un alphabet ce qui permet de distinguer les littéraux explicites des littéraux implicites.

Exemple 3 :

Pour la requête précédente on obtient la suite de règles :

$$W(b_2) = x_1 W(a_2) \cup x_3 W(b_3)$$

$$W(c_3) = x_2 W(a_2) \cup x_4 W(b_3)$$

$$W(a_2) = a_1$$

$$W(b_3) = b_1 W(b_2)$$

$$Q = c_1 c_2 W(c_3)$$

Par substitutions on obtient les règles simplifiées suivantes:

$$W(b_2) = x_1 a_1 \cup x_3 b_1 W(b_2)$$

$$Q = c_1 c_2 x_2 a_1 \cup c_1 c_2 x_4 W(b_2)$$

Le problème est maintenant de s'affranchir du terme défini récursivement ie $W(b_2)$.

Pour cela introduisons quelques définitions.

Définition 1 :

Soit un alphabet $A = \{a_1, \dots, a_n\}$. Les expressions régulières sur cet alphabet sont données par:

- (i) $a_1, \dots, a_n, k, 0$
- (ii) si P et Q sont régulières alors $P \cup Q, PQ, P^* = k \cup P \cup P^2 \cup \dots$, le sont aussi.
- (iii) toutes les expressions régulières sont générées par les règles (i) et (ii).

Définition 2 :

Soit q une fonction définie sur les expressions régulières par la suite de règles suivantes:

- (i) $q(a_i) = 0$
- (ii) $q(0) = 0$
- (iii) $q(k) = 1$
- (iv) $q(P \cup Q) = 0$ si $q(P)$ et $q(Q) = 0$
 $= 1$ sinon
- (v) $q(PQ) = 1$ si $q(P)$ et $q(Q) = 1$
 $= 0$ sinon
- (vi) $q(P^*) = 1$

Theoreme 1 :

L'équation $D = a \cup bD$ ou a et b sont des expressions régulières et $q(a) = 0$ a pour solution : $D = b^*a$.

Remarque:

Ce théorème n'est autre qu'un théorème de point fixe sur un alphabet et permet donc de passer d'une représentation inductive d'ensemble à une représentation explicite.

Les notions nécessaires de théorie des langages, ainsi que la démonstration du théorème peuvent être trouvés dans [Brzo66].

Ce théorème appliqué à l'exemple précédent donne pour Q l'expression suivante:

$$Q = c_1 c_2 x_2 a_1 \cup c_1 c_2 x_4 b_1 (x_3 b_1)^* x_1 a_1$$

c'est à dire en remplaçant les x_i par les littéraux concernés:

$$Q = c_1 c_2 (c_3 a_2) a_1 \cup c_1 c_2 (c_3 b_3) b_1 ((b_2 b_3) b_1)^* (b_2 a_2) a_1$$

Q s'écrit donc comme la réunion de deux expressions régulières;

le premier terme ne posant aucune difficulté, [Chang78], considérons uniquement le deuxième.

Par substitution et renommage des variables on obtient les expressions suivantes:

c_1	EMPLOYES(*nom = s, dept = t)
c_2	VENTE(dept = t, objet = fleur)
(c_3)	(COMMANDE(superieur = jean, nom=s))
(b_3)	(COMMANDE(superieur = w, nom = u))
b_1	EMPLOYEE(nom = u, manager = v)
$((b_2))$	((COMMANDE(superieur = w, nom = v))*)
$((b_3))$	((COMMANDE(superieur = ww, nom = uu))
(b_1)	(EMPLOYES(nom=uu, manager = vv)
b_2	COMMANDE(superieur = ww, nom = vv)


```

a2      COMMANDE(superieur = x,nom = y)
a1      EMPLOYES(nom = y,manager = x)

```

Après unification des termes compris entre parentheses:

```

EMPLOYES(*nom = s,dept = t)
VENTES(dept = t,objet = fleur)
EMPLOYES(nom = s,manager = v)
EMPLOYES(nom = v,manager = vv)*
EMPLOYES(nom = vv,manager = jean)

```

Cette suite de commandes conduit alors naturellement au programme suivant:

```

EVALUATE EMPLOYES(*nom = vv,manager = jean);
*V <--- *VV
DO WHILE *V = empty
    EVALUATE (EMPLOYES(nom = s,manager = v)&
              EMPLOYES(*nom = s,departement = t)&
              VENTES(departement = t,objet = fleur)) for each v E *V
    PRINT *S
    EVALUATE EMPLOYES(*nom = v,manager = vv) for each vv E *VV
    *VV <--- *V
END;

```

Remarque:

Cette méthode s'applique seulement à une classe d'axiomes rékursifs, classe dite régulière, c'est à dire ne comportant au plus qu'une relation virtuelle dans leur membre de gauche.

Il est cependant possible d'étendre cette classe d'axiomes, [Mink83], à ceux qui comportent plusieurs relations non implantées dont au plus une n'est pas résolvable indépendamment de cette clause.

Exemple 4 :

Soit l'ensemble de règles suivantes:

(i)

$F \rightarrow A$

$F \& A \rightarrow A$

(ii)

$P \rightarrow N$

$A \& N \rightarrow N$

ou les relations P et N sont des relations implantées.

Il suffit alors pour résoudre (ii) d'avoir résolu le système (i); par ce type de remarque on peut ainsi enrichir la classe d'axiomes rékursifs que l'on peut traiter avec la méthode de Chang et ainsi traiter certains types de rékursivités croisées.

2.3 Méthode de Henschen et Naqvi

Les auteurs proposent dans [Hens83] un algorithme qui à partir d'une requête exprimée sous forme de point fixe la transforme en un programme itératif d'opérations de l'algèbre relationnelle.

Cet algorithme est détaillé dans [Hens84].

Rappel sur la sémantique du point fixe.

L'utilisation de la programmation logique et en particulier des clauses de Horn comme langage de définition et d'interrogation de bases de données relationnelles a conduit certains auteurs à développer une classification des requêtes exprimables dans les langages d'interrogation.

Il a été démontré dans [Chan82a] et [Chan82b] que l'algèbre relationnelle ne permettait pas d'exprimer des requêtes du type point fixe et que la puissance d'expression d'un langage basé sur des clauses de Horn est exactement celle procurée par l'application d'un opérateur de point fixe à des formules positives du premier ordre.

De telles requêtes apparaissant naturellement lors de l'interrogation d'une base de données, il est souhaitable de pouvoir enrichir l'algèbre relationnelle de ce type d'opérateur [Aho79], et ainsi de préserver l'aspect non procédural du langage d'interrogation.

Une autre optique choisie par les auteurs de cet article est de simuler cet opérateur par un ensemble de procédures comportant une suite de commandes relationnelles classiques.

Définition 3 :

Soit une expression du type $R=f(R)$ (1)

ou R est une relation et f la composée d'opérateurs de l'algèbre relationnelle.

On appelle plus petit point fixe de l'équation (1) une relation R^* qui vérifie:

$$(i) \quad R^* = f(R^*)$$

$$(ii) \quad \text{si } P \text{ est une relation telle que } P = f(P)$$

$$\text{alors } R^* < P$$

Theoreme 2 (Tarski) :

Soit une fonction vérifiant:

$$R_1 < R_2 \implies f(R_1) < f(R_2)$$

alors cette fonction est dite monotone et l'équation

$$R = f(R) \text{ possède un point fixe unique.}$$

Une démonstration de ce théorème peut être trouvée dans [Apt82].

Exemple 5 :

La fermeture transitive d'une relation binaire R_0 est le point fixe de l'équation $R = f(R)$ avec $f(R) = P_1 \circ_4 (s_2 \circ_3 (R \times R_0))$.

La recherche d'une relation vérifiant une équation de type (1) peut se résumer en spécifiant les deux étapes suivantes:

$$(2) \quad \text{initialisation} \quad R_0 < R$$

$$\text{induction} \quad f(R) < R$$

Remarquons alors qu'une simple boucle "while" permet d'exprimer une procedure synthetisant la recherche d'un point fixe.

```
(3)  R <--- Ro
      DO
        R' <--- R
        R <--- R U f(R)
      WHILE (R' <> R)
```

Les travaux de Henschen et Naqvi consistent donc à partir d'un ensemble d'axiomes de type (2) à constituer un programme itératif de type (3) et ceci en profitant des optimisations possibles procurées par la requete.

Leurs propositions peuvent se résumer en deux étapes:

- la premiere consiste à utiliser un graphe de connection similaire à celui qu'utilise Chang pour réécrire un ensemble de clauses récursives sous forme de clauses ou la récursion ne porte que sur un seul prédicat.
- la seconde produit à partir d'une clause réguliere un programme de type (3). Ces programmes seront alors stockés sous forme précompilée et seront appelés et exécutés lors de l'évaluation d'une requete.

Comme nous le préciserons plus tard le stockage de ces programmes peut devenir problematique puisqu'a un ensemble de clauses régulières peut être associé plusieurs programmes correspondant à des requetes

différentes.

Developpons sur un exemple l'étape 2;

Soient les clauses suivantes désormais célèbres:

$PERE(x,y) \text{ ---> } ANCETRE(x,y)$

$ANCETRE(x,y) \& PARENT(y,z) \text{ ---> } ANCETRE(x,z)$

et la requête $ANCETRE(x,a)$

Les techniques de chaînage arrière conduisent alors à l'algorithme suivant:

1. -retrouver $PERE(w,a)$ et mettre les tuples résultats dans la relation $ANCETRE$
2. -"empiler" la constante 'a'
3. -"dépiler" une valeur de la pile et en affecter la variable z
4. -retrouver $PARENT(y,z)$ et utiliser les valeurs de 'y' pour retrouver $PERE(w,y)$, dépiler les valeurs de y;
insérer les tuples résultats dans la relation $ANCETRE$
5. -aller en 3

Considérons maintenant la requête $ANCETRE(a,y)$.

Il est alors clair que l'algorithme précédent ne s'applique pas à cette requête:

la requête ne provoque en effet aucune instanciation dans la relation $PARENT$ et ne permet donc pas de poursuivre le processus de résolution tel qu'il a été défini ci dessus, (cf les limitations des techniques de

chainage arriere). Il est alors necessaire d'instancier le predicat ANCETRE figurant dans la deuxieme clause à l'aide des résultats procurés par la résolution de la premiere clause afin de pouvoir résoudre le predicat PARENT.

L'algorithme doit etre donc légèrement modifié.

De maniere générale, considerons des clauses régulières du type suivant:

(i) $F1 \& F2 \& \dots \& F_n \quad \text{---} \rightarrow R$

(ii) $A1 \& A2 \& \dots \& R \& B1 \& \dots \& B_n \quad \text{---} \rightarrow R$

ou l'on appellera $F1 \& F2 \& \dots \& F_n$ la partie déterminée de l'ensemble de clauses et $A1 \& A2 \& \dots \& R \& B1 \& B2 \& \dots \& B_n$ la partie induite.

En se reportant à l'exemple précédent on remarque alors que les instantiations procurées par la requete influent sur l'ordre dans lequel doivent etre utilisées les parties déterminées et induites. Cet ordre peut etre défini comme suit:

si le sous-but $A1 \& \dots \& R \& \dots \& B_n$ est résolvable, ie comporte au moins une variable instanciée, alors l'ordre est : évaluer PI, puis PD
sinon évaluer PD puis PI.

Soit donc l'algorithme général: Algorithme 3

```

R = R U eval(PD)
insérer-dans-pile P (constantes issues de la requete)
DO WHILE P <> 0
  Z <--- contenu-de-la-pile P
  Y = eval(PD ou PI, Z)
  R = R U eval(PI ou PD, Y)
  insérer-dans-pile P (nouvelles constantes)
OD

```

Remarques:

Cette approche autorise, à la différence de celle de Chang, la compilation d'axiomes comportant plusieurs prédicats récursifs, reductibles à des axiomes réguliers.

Le contrôle de ces récursivités multiples présente de notables difficultés et conduit à des algorithmes particulièrement complexes, les auteurs en laissant d'ailleurs la résolution aux chercheurs "more concerned in operating systems". Il est à noter que pour les récursions portant sur une seule variable les deux méthodes sont identiques.

Les complexités relatives des deux méthodes ont été étudiées dans [Viell84].

La méthode de Henschen et Naqvi s'avère être plus performante en présence de récursivité portant sur plusieurs variables, les conditions d'arrêt portant généralement sur des ensembles de taille plus réduite. Il apparaît également que le degré de connectivité des graphes représentant les relations explicites est un facteur influençant fortement les performances relatives de ces deux méthodes.

2.4 Méthode d'Alexandre.

L'originalité de cette méthode consiste à utiliser conjointement les techniques de chaînage arrière et de chaînage avant. L'idée directrice étant de déterminer en chaînage arrière les règles pertinentes qui en chaînage avant répondront à la requête.

La solution proposée par les auteurs [Bull85] peut se décomposer en les étapes suivantes:

1. La requête est considérée comme un but à évaluer en chaînage arrière.
2. Cette requête est traduite en un ensemble de clauses qui seront exécutées en chaînage avant.
3. Cet ensemble de clauses est traduit en une suite d'opérations de l'algèbre relationnelle.

Les auteurs dans cet article ne s'intéressent qu'au passage de l'étape 1 à l'étape 2, le passage de l'étape 2 à l'étape 3 ayant été étudié puis réalisé dans le cadre d'un moteur d'inférence appelé BOUM, [Jamie83].

Nous nous proposons de traiter un exemple désormais classique ce qui permettra d'illustrer la spécificité de cette méthode.

Soient donc les clauses suivantes:

$PERE(x,y) \rightarrow ANCETRE(x,y)$

$PERE(x,y) \& ANCETRE(y,z) \rightarrow ANCETRE(x,z)$

La requête étant par exemple: $ANCETRE(a,x)$.

Ce but est alors coupé en deux, un problème et un ensemble de solutions, ce qui permettra de simuler le chaînage arrière par les techniques de chaînage avant.

Le problème s'écrit sous la forme d'un prédicat de la forme $Pb-Ancetre(a)$ et une solution est un fait du type $Sol-Ancetre(a,b)$.

A l'aide de ces deux prédicats il est possible d'écrire les règles qui permettront de générer les problèmes et les solutions en chainage avant:

Les deux axiomes initiaux sont alors transformés en:

$$1 \text{ Pb-Ancetre}(x) \& \text{PERE}(x,y) \text{ ---> Sol-Ancetre}(x,y)$$

ce qui peut s'interpréter par: si l'on a le problème de trouver les descendants de x et que x est le père de y alors y est une solution.

$$2 \text{ Pb-Ancetre}(x) \& \text{PERE}(x,y) \& \text{ANCETRE}(y,z) \text{ ---> Sol-Ancetre}(x,z)$$

ce qui signifie: si l'on a le problème de trouver les descendants de x , que x est le père de y et que y est l'ancêtre de z , alors z est une solution.

Cette dernière règle comportant elle même un but, ce dernier subit la même transformation que précédemment ce qui conduit à réécrire la règle 2 avec les deux règles suivantes:

$$2.1 \text{ Pb-Ancetre}(x) \& \text{PERE}(x,y) \text{ ---> Pb-Ancetre}(y)$$

$$2.2 \text{ Sol-Ancetre}(y,z) \text{ ---> Sol-Ancetre}(x,z)$$

On remarque alors que la règle 2.2 est incorrecte, la variable x ne figurant pas dans la prémisse de la clause. Il faut pouvoir spécifier que cette variable provient de la clause 2.1. A cet effet il est introduit un prédicat supplémentaire, dit de contexte, qui préserve cette information.

Les deux règles s'écrivent alors:

2.1' Pb-Ancetre(x)&PERE(x,y) ---> Pb-Ancetre(y)&Cont(y,x)

2.2' Sol-Ancetre(y,z)&Cont(y,x) ---> Sol-Ancetre(x,z)

En rajoutant la règle 1 on obtient alors un ensemble de règles qui en chainage avant répondront à la requête posée.

Remarques:

Cette méthode permet la compilation d'axiomes non réguliers, ce qui autorise par exemple la définition de la relation ANCETRE par des axiomes du type

ANCETRE&ANCETRE ---> ANCETRE.

Il reste cependant à gérer les processus de transformation ainsi que les prédicats de contexte pour des récursivités multiples ce qui devrait alourdir sensiblement les techniques à mettre en oeuvre.

De plus rappelons que les règles de productions obtenues par cette méthode doivent être traduites en algèbre relationnelle avant de pouvoir être mises en oeuvre sur un SGBD.

Une solution compilée aux problèmes des axiomes récursifs a été également proposée par Marque-Pucheu [Marq83]. Une formalisation du problème est proposée et des solutions partielles sont présentées.

Il s'agit d'une méthode purement algébrique basée sur une extension de la notion d'opérateurs relationnels et sur les automates d'états finis. Il est montré que pour une suite d'opérations relationnelles donnée (représentée par un arbre ou un ensemble d'arbres) il existe toujours un arbre optimisé équivalent.

De manière pratique il est nécessaire de recenser un ensemble de règles d'optimisation afin de développer des heuristiques de compilation.

3. Les méthodes interprétées.

3.1 Introduction.

Dans cette approche, les processus de résolution et de recherche des informations dans la base de faits sont intimement liés. La réponse à une requête mettra en jeu de façon simultanée les axiomes et les faits impliqués.

Cette méthode provenant des techniques utilisées en Intelligence Artificielle est similaire à celle suivie dans les "démonstrateurs de théorèmes" et est également celle adoptée par le langage PROLOG.

Le problème initial est décomposé par chaînage arrière en sous problèmes qui seront résolus par des accès successifs à la base de données et dont la réunion des réponses fournies formera la réponse complète à la requête.

Afin de pouvoir utiliser cette méthode dans le contexte des bases de données déductives il a été proposé deux types d'optimisations :

- utilisation d'une fonction de sélection qui indique lors de la résolution d'une clause conjonctive l'ordre dans lequel ils seront évalués [Mink78a, Mink78b].

- regroupement des accès concernant une sous-requête afin de pouvoir obtenir les solutions de manière ensembliste [Mink75, Mink78b].

Notons que les problèmes attachés au traitement des axiomes récurifs persistent pour l'approche interprétée, ces problèmes étant liés au mécanisme d'inférence utilisé, le chaînage arrière, et non à l'ordonnancement des processus informatiques mis en oeuvre.

3.1 Méthode de Lozinskii.

Cette méthode, [Lozin84], [Lozin85], consiste à partir de la requête initiale à déterminer de manière dynamique les sous-problèmes engendrés et les faits nécessaires à la résolution de ces sous-problèmes. Ces derniers seront résolus "at run time" et leur réunion formera l'ensemble des solutions demandées. Le problème consiste donc pour chaque sous requête à déterminer ce qui est appelé "faits pertinents", puis s'ils existent, d'engager en chaînage avant le processus de recherche des informations élémentaires dans la base de données.

Cette approche combine donc la technique de chaînage arrière pour décomposer la requête en sous-questions et pour déterminer les faits pertinents et le chaînage avant pour évaluer ces sous-questions sur la base de données.

Ceci permet donc de s'affranchir théoriquement des problèmes de conditions d'arrêt, et autorise également l'arrêt anticipé du processus engagé en cas de non existence de faits pertinents à la requête posée.

De manière classique, les axiomes pertinents de la requête sont représentés par un graphe ET/OU, où les ET-noeuds représenteront les relations mises en jeu et les OU-noeuds les axiomes proprement dit.

Dans le cas où une relation est définie de manière récursive, l'occurrence de celle-ci apparaissant dans la prémisse d'une clause sera représentée par un noeud principal et celle apparaissant en partie droite par un noeud secondaire.

Exemple 5 :

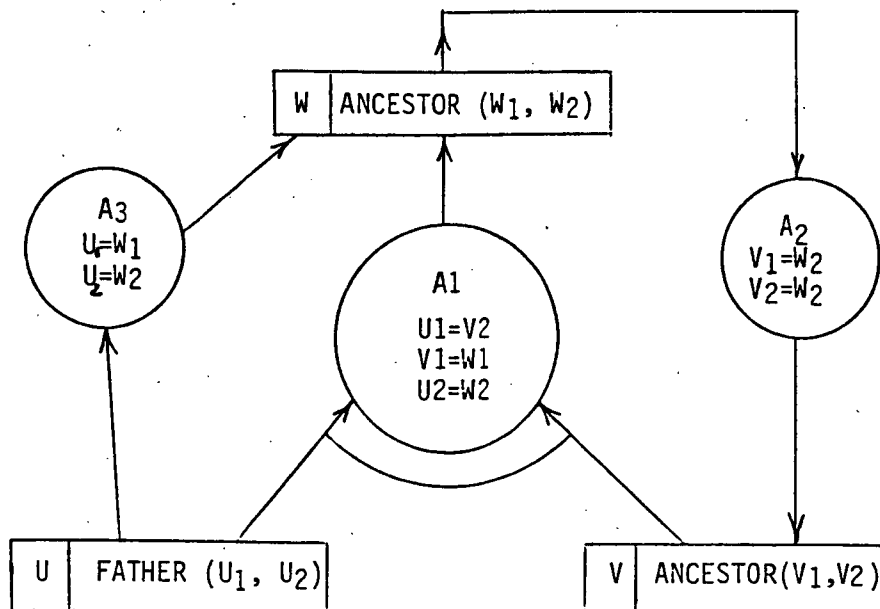
la relation ANCESTRE pourra être définie par les axiomes suivants:

$PERE(u1/nom, u2/nom) \& (u1=w1) \& (u2=w2) \dashrightarrow ANCESTRE(w1/nom, w2/nom)$

$ANCESTRE(x1/nom, x2/nom) \& PERE(y1/nom, y2/nom) \& (x2=y1) \& (x1=z1) \& (y2=z2)$

$\dashrightarrow ANCESTRE(z1/nom, z2/nom)$

et cette définition conduira au graphe suivant:



Schema 5

Recherche des faits pertinents.

Définition 4 :

Assurons que toutes les variables apparaissant dans les clauses pertinentes d'une requête portent des noms différents.

Soit alors l'ensemble $MIG(x)$ défini comme suit:

(i) S'il existe un axiome,

$$Q(\dots) \& \dots \& P(\dots, x, \dots) \& \dots \& (x=y) \& \dots \rightarrow R(\dots, y, \dots)$$

alors $y \in MIG(x)$.

(ii) Si $y \in MIG(x)$ et $z \in MIG(y)$ alors $z \in MIG(x)$.

Cet ensemble représente donc les différents termes vers lesquels x peut "migrier" lors du processus de résolution.

Remarquons que cet ensemble peut être aisément calculé en parcourant un graphe du type de celui qui a été défini précédemment. Une procédure est d'ailleurs proposée à cet effet dans l'article étudié.

Définition 5 :

Soient une requête $R: R(x_1, x_2, \dots, x_i = a_i, \dots)$ et un fait

$P(y_1 = b_1, \dots, y_j = b_j, \dots)$.

P est dit pertinent de Q si P contient y_j tel que $x_i \in MIG(y_j)$

et $b_j = a_i$.

Exemple 6 :

Pour la relation ANCETRE on a :

$MIG(x1) = \{x1, y1, z1\}$ et $MIG(x2) = \{x2, y2, z2\}$

Soient les requetes $Q1:ANCETRE(x,a)$ et $ANCETRE(a,y)$.

Les faits relevant de $Q1$ sont les faits du type: $PERE(z,a)$

et ceux relevant de $Q2$: $PERE(a,z)$.

Nous pouvons alors énoncer l'algorithme general propose par Lozinskii:

Soient Q la requete considerée, F l'ensemble des faits appartenant à l'EDB.

Algorithme 3 :

1. Selectionner les faits relevant FR de la requete Q .
2. S'il existe un axiome executable sur FR
 alors executer et rajouter les faits nouveaux dans FR .
 sinon rechercher dans F les faits necessaires à l'execution
 et rajouter les faits nouveaux dans FR .
3. Aller en 2 tant que des faits nouveaux sont produits sinon FIN.

Exemple 7 :

Soient les relations M, P, R et les relations virtuelles N et S définies par les axiomes suivants:

$A1: S(s1, s2) \& P(p1, p2) \& (s2=p1) \& (s1=n1) \& (p2=n2) \text{ ---> } N(n1, n2)$

$A2: M(m1, m2) \& N(n1, n2) \& (m2=n1) \& (m1=s1) \& (n2=s2) \text{ ---> } S(s1, s2)$

$A3: R(r1, r2) \& (r1=n1) \& (r2=n2) \text{ ---> } N(n1, n2)$

Soient par exemple les extensions suivantes des relations M, P, R:

M	P	R
-----	-----	-----
m1 m2	p1 p2	r1 r2
-----	-----	-----
b c	d a	e d
c e	g h	t g
f t		

Soit la requete $Q: S(x, a)$.

Appliquons l'algorithme précédent à cet exemple.

1. $P(d, a)$ est le seul fait relevant de Q.
2. Pour executer A1 il faut un fait du type $S(x1, d)$ pour effectuer la jointure. Pour obtenir un tel fait résolvons $Q2: S(x1, d)$.
 $R(e, d)$ est relevant de Q2 (car $s2 \in \text{MIG}(r2)$).
 L'execution de A3 sur $R(e, d)$ fournit $N(e, d)$.
 Pour executer A2 sur $N(e, d)$ il faut un fait du type $M(x2, e)$.
 Résolvons donc $Q3: M(x2, e)$, ce qui fournit pour solution $M(c, e)$.
 A2 genere $S(c, d)$ et A1 $N(c, a)$. Il faut donc un fait $M(x3, c)$
 ce qui conduit à la sous requete $Q4: M(x3, c)$ qui donne
 pour solution $M(b, c)$.
 Finalement A2 genere l'unique réponse $S(b, a)$.
3. FIN.

Remarques:

Cet algorithme est naturellement développé dans l'article étudié et

des preuves de sa complétude et de sa terminaison sont proposées. Il est donné diverses optimisations en vue d'une implémentation possible afin par exemple de générer les réponses de manière ensembliste et non "tuples à tuples".

Il est également proposé des techniques d'implémentations parallèles des processus d'inférence.

Notons que cette méthode s'applique à toute classe d'axiomes récurrents. Cette solution est à rapprocher de celle proposée dans [Bull85] et peut être considérée comme une version interprétée de cette dernière.

Une autre solution interprétée est proposée par McKay et Shapiro [McKay81].

Le formalisme utilisé repose sur les "active connection graph" dans lesquels les arcs relient les prédicats unifiables. Cette méthode n'effectue aucune distinction entre l'EDB et l'IDB et semble donc être plus particulièrement destinée aux applications de l'intelligence artificielle.

Elle n'effectue aucune restriction sur les axiomes, mais ne permet pas de donner la priorité aux opérations de sélection : il est par exemple nécessaire de calculer la totalité de la relation Ancêtre afin de répondre à une requête du type Ancêtre(toto,y).

Il reste que les techniques proposées de part leur aspect interprété ne semblent pas pouvoir bénéficier pleinement des performances proposées dans le domaine de l'optimisation de requêtes par les SGBD relationnels.

4. Autres méthodes.

4.1 Méthodes de Reiter et de Minker, Nicolas.

Les deux méthodes qui sont étudiées dans ce paragraphe permettent de s'affranchir des problèmes liés aux axiomes récursifs en séparant de façon "adequate" l'EDB de l'IDB.

Il s'agit de rendre explicites les relations définies par des axiomes récursifs pouvant mener à des chemins de dérivation potentiellement infinis.

Soit l'ensemble de clauses de Horn suivant:

$P1 \& C1 \rightarrow P'2$

$P2 \& C2 \rightarrow P'3$

$P3 \& C3 \rightarrow P'4$

• •
• •
• •

$Pn \& Cn \rightarrow P'1$

ou pour chaque i P_i et P'_i sont des littéraux unifiables de l'EDB. Un tel système initialisé avec l'un des P_i conduit à parcourir des chemins de longueur infini.

L'idée est alors de couper ce chemin de dérivation en rendant l'une des règles improductive lors du processus de dérivation.

La méthode de Reiter [Reit78b], consiste à rendre l'un des L_i

"extensionnellement complet" vis à vis d'une des règles R_i ou il figure, ie, tous les faits dérivables par R_i figurent déjà dans la relation L_i . La règle R_i est alors considérée comme contrainte d'intégrité pour assurer cette propriété lors des mises à jours sur la base de données, [Nico78].

Minker et Nicolas [Mink83], proposent d'utiliser cette règle en génération.

Ces deux méthodes rendent donc explicites des faits définis implicitement, soit de manière statique en utilisant l'intégrité, soit de manière dynamique en alternant génération et dérivation.

Ces solutions s'appliquent à toutes les classes d'axiomes récursifs mais altèrent sérieusement l'intérêt de définir des faits de manière implicite.

Remarque :

Dans la méthode proposée par Minker et Nicolas, une relation pouvant être définie conjointement par des règles de dérivation et des règles de génération, il est nécessaire afin d'éviter toute perte d'information, d'activer ces dernières lors de "mises à jours virtuelles" effectuées par les règles de dérivation, [Mink83].

4.2 Méthodes de reconnaissance syntaxique.

Le but de ces méthodes ,[Mink83] ,[Ioan85] ,est de déterminer "at compile time " une condition de terminaison du processus d'évaluation qui soit indépendante du contenu de la base de donnée.

Cette condition est déterminée par une étude syntaxique des prédicats récursifs et plus particulièrement de la position des variables qui y figurent.

Il est ainsi exhibé des classes de prédicats définis récursivement sur lesquels on peut "à priori" fixer une borne supérieure à la profondeur du graphe de résolution.

Ces méthodes sont donc partielles et sont liées à un mécanisme de résolution en chainage arriere.

Elles nécessitent de plus une étude syntaxique qui peut s'avérer couteuse.

En résumé ,elles ne semblent pas etre adaptées à un contexte de bases de données déductives.

5. Conclusion.

Les diverses solutions étudiées sont pour la plupart des améliorations des techniques utilisées en Intelligence Artificielle ; les algorithmes proposés transforment une requête portant sur une relation définie récursivement en suite de requêtes plongées dans un langage procédural de programmation (méthode de compilation) , ou en une suite d'appels au SGBD (méthode inter

Le langage de manipulation est du type PROLOG et ne permet donc pas aisement d'inclure des fonctions spécifiques aux bases de données du type agrégats ; de plus les méthodes d'inférences utilisées sont basées sur le principe de résolution de Robinson.

Ces méthodes sont donc particulièrement adaptées au couplage entre un moteur d'inférence et un SGBD classique.

Une autre approche possible serait d'enrichir l'algèbre relationnelle d'opérateurs de point fixe , afin de transformer les clauses récursives directement en une suite d'opérations relationnelles.

Ceci permettrait une meilleure optimisation globale du processus d'inférence ainsi que la possibilité de développer des algorithmes performants de "jointures répétitives" liées à cette opération , [Vald85].

Dans cette optique , il est alors nécessaire de développer des techniques locales d'optimisation de requêtes récursives , telles que par exemple la priorité à donner aux sélections par rapport aux jointures dans un opérateur de point fixe.

En effet ,s'il est aisé ,avec l'approche interprétée , de tirer parti des constantes générées par la requete ,aucune solution satisfaisante n'a encore été proposée dans cette optique.

Il reste également à définir une mesure de complexité permettant de comparer efficacement les diverses méthodes publiées.On peut trouver dans [Banc85] une ébauche de solution à ce probleme.

BIBLIOGRAPHIE :

- Apt82 K.R.Apt ,M.H.Van Emden.
Contributions to the Theory of Logic
Programming.
J.ACM ,Vol 29, No 3 ,Juillet 82.
- Aho79 A.V.Aho,J.D.Ullman.
Universality of Data Retrieval Languages
Conf on POPL San-Antonio,Texas,79.
- Banc85 F.Bancilhon.
Naive evaluation of recursively defined
predicates.
Proc of Islamadora Workshop on Database
and AI. Mars 85.
- Bull85 Une solution pour traiter les axiomes
recursifs dans les bases de donnees
deductives.
Rapport de recherche DRAL/IA/45.01.
- Brzo66 J.A.Brzozowski.
Class Notes of Switching and Automata
Theorie ,Dept. of Electrical Engineering
and Computation Science .
University of California ,Berkeley.
- Chand82a A.K.Chandra,D.Harel.
Horn Clauses and the Fixpoint
Query Hierarchy.
Proc ACM Symp 82.
- Chand82b A.K.Chandra,D.Harel.
Structure and Complexity
of Relational Queries.
Journal of Computer and
System Sciences 25 1982.

Chang78

C.Chang.
 DEDUCE 2 :further investigations of
 deduction in relational data bases.
 In Logic and Data Bases.
 Edited by H.Gallaire and J.Minker
 Plenum NewYork.

Chang81

C.Chang.
 On evaluation of queries containing derived
 relations in a relationnal data base.
 In Advances in Data Base Theory
 Edited by H.Gallaire,H.Minker and JM.Vicolas
 Vol 1 Plenum Press , New York.

Chang73

Chang and Lee
 Symbolic Logic and Mechanical Theorem Proving
 Academic Press , New York.

Hens83

L.J.Henschen,S.A.Naqvi.
 Synthesizing Least Fixed Point Queries
 into Non-recursive Iterative Programs.
 Proc of 8th IJCAI 83.

Hens84

L.J.Henschen,S.A.Naqvi.
 On Compiling Queries in Recursive First-Order
 Databases.
 JACM Vol 31,No 1, Jan 84.

Ioan85

Y.E Ioannidis .
 A time bound on the materialization
 of some recursively defined views.
 Proc of VLDB 219-225. Aout 1985.

Jamie83

M.Jamier,J.M.Pugin.
 Le moteur d'inference BOUM.
 Rapport de DEA.
 Ecole Centrale de Paris ,juin 83.

Kain

R.Kain.
 Automata Theory : Machines and Languages
 Mc Graw-Hill computer science series.

- Kowal75 R.Kowalski.
A proof procedure using connection graph.
JACM 22,4 Oct 75.
- Lozin84 E.L.Loizinskii.
Inference by generating and structuring
of deductive databases.
Technical Report 84-11 ,Hebrew university
Department of computer science ,juin 84 .
- Lozin85 E.L.Loizinskii.
Evaluating queries in Deductive Databases
by Generating.
IJCAI , 173-177. Los Angeles ,Aout 85.
- Marq83 G.Marque Pucheu.
Algebraic structure of answers in a
recursive logic database.
Rapport Ecole normale superieure.
- McKay81 D.McKay,S.Shapiro.
Using connection graphs for reasoning with
recursive rules.
Proc 7th IJCAI,1981.
- Mink75 J.Minker.
Performing inferences over relational
data bases.
ACM SIGMOD. San Jose ,California ,mai 75.
- Mink78b J.Minker.
An experimental relational database system
based on logic.
Logic and Data bases,H.Gallaire,J.Minker
Plenum Press,New-York 1978.
- Mink78a J.Minker.
Search Strategy and selection function
for an inferential relational system.
ACM Trans.Database Syst 3,1 .Mar 78.
- Mink83 J.Minker,J.M.Nicolas.
On recursive axioms in deductive data bases
Information Systems 8.

- Nico78 J.M.Nicolas , K.Yazdanian.
Integrity checking in deductive databases.
Logic and Data bases, H.Gallaire,J.Minker
Plenum Press ,New-York 1978.
- Nico83 J.M.Nicolas , K.Yazdanian.
An Outline of BDGEN:A Deductive DBMS.
IFIP 83.
- Reit78a R.Reiter.
Deductive question answering on relational
databases.
In Logic and Data Bases.H Gallaire,J.Minker.
Ed Plenum,New York.
- Reit78b R.Reiter.
On structuring a first-order database.
Proc of the 2nd Canadian Society
for Computer Science
National Conference,Canada July.
- Ullm85 J.D.Ullman.
Implementation of Logical Query Languages
for Databases.
ACM SIGMOD 1985.
- Vieil84 L.Vieille.
La compilation d'axiomes recursifs.
Rapport de stage de DEA.
Universite Paris VI ,oct 84.
- Vald85 P.Valduriez.
Optimisation des operateurs relationnels
dans une machine base de donnees.
These d'Etat.
Universite Paris 6 ,1985.

